LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-635646

Fast MAP Search for Compact Additive Tree Ensembles (CATE)

R. J. Prenger, B. Y. Chen, T. L. Marlatt, D. M. Merl

April 23, 2013

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Fast MAP Search for Compact Additive Tree Ensembles (CATE)

R. J. Prenger, B. Y. Chen, T. L. Marlatt, and D. M. Merl
Lawrence Livermore National Laboratory

March 21, 2013

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

1 Introduction

Since their inception, additive ensembles have been shown to provide a highly effective and robust framework for a variety of classification and prediction tasks. Their successful application in a practical setting, however, usually depends on specifying a priori several operational parameters that can be difficult or time-consuming to estimate. In particular, the number of constituent models (e.g., decision trees, logistic models) that comprise the ensemble induces a tradeoff between performance and efficiency that may not be readily characterized a priori. This dichotomy varies with several key factors (e.g., the nature of the underlying models, availability of training data, etc.) and is especially critical in contexts where memory is at a premium (e.g., limited hardware [19]) and/or computational speed is paramount (e.g., real-time stock trading, cybersecurity).

We have frequently observed that the classification performance of the ensemble tends to plateau (i.e., achieve *saturation*) as the number of members in the ensemble (i.e., ensemble size) increases, so that eventually the marginal gain from adding further models falls below a desired tolerance. This observation suggests that superior performance may be achieved via overspecification of the ensemble size. In practice, however, this strategy can easily produce simultaneously redundant and overly complex ensembles at a significant expense to computational efficiency.

To address this problem, we present a greedy maximum a posteriori (MAP) search algorithm that combines the tree-based ensemble concept with Bayesian principles to enable more efficient training and inference within the ensemble framework. Our approach utilizes a novel formulation of the full prior distribution, in which the number of component models is regularized by incorporating the parameters required for each model, creating a penalty for complexity. The algorithm then employs boosting to optimize the joint pos-

terior distribution induced by this model formulation. As a result, the decision tree associated with each successive model becomes progressively simpler so that eventually no further component models are added to the ensemble. Thus, in a departure from existing methods, our approach allows the ensemble size parameter to be estimated from the data without the need for costly processing tools such as reversible jump MCMC [7]. The resulting Compact Additive Tree Ensemble (CATE) operates with reduced redundancy and complexity, facilitating improvements in both computational and memory efficiency.

The remainder of this paper is organized as follows. Section 2 contains a review of the existing literature on additive tree models, with an emphasis on the connections between formal Bayesian methods and Random Forest [3], the cornerstone of tree-based ensembles. Section 3 provides the details of our tree-based classification model, including the full prior specification. Section 4 describes the greedy MAP search algorithm used for finding the optimal tree model of Section 3. In Section 5 we present empirical results comparing the performance of the compact ensemble with the Random Forest (RF) and Bayesian Additive Regression Trees (BART). We show that the compact ensembles produced by our approach achieve comparable accuracy with significantly reduced memory requirements and increased prediction throughput rates. Section 6 concludes with a brief discussion.

2 Additive Tree Ensembles

In the additive tree ensemble framework, the functional relationship between an outcome y and its set of associated features $x = [x_1, x_2, \ldots, x_K]$ is modeled via an additive combination of simple predictive models. Each of these models is characterized by a partition of a multidimensional feature space represented by a decision tree $T_i \in \{T_1, T_2, \ldots, T_M\}$ and its associated parameters $\theta_i = [\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,P_i}]$, where M

is the number of models and P_i is the number of partitions (i.e., leaf nodes) in T_i. Thus for a given tree T_i, y is related to x via the parameters $\theta_{i,p_i(x)}$ associated with the leaf node $p_i(x)$ to which x belongs. For example, linear regression can be accomplished using a sum of tree models, given by

$$\begin{aligned} y_t|x_t &= f(x_t) + \epsilon_t \\ f(x_t) &= \sum_{i=1}^{M} f(x_t; T_i, \theta_i) \end{aligned}$$

where ϵ is a residual noise term, M is the number of tree models comprising the ensemble, and $f(x_t; T_i, \theta_i)$ is the outcome assigned to x_t given the tree T_i and its leaf parameters θ_i. Similarly, nonlinear classification can be accomplished via a standard probit or logit GLM in which the continuous latent feature is regarded as above:

$$\begin{aligned} p(y_t = 1|\eta_t) &= \text{logit}^{-1}(\eta_t) \\ \eta_t|x_t &= \sum_{i=1}^{M} f(x_t; T_i, \theta_i) \end{aligned}$$

where

$$\text{logit}^{-1}(x) = \frac{1}{1 + \exp(-x)}$$

In general, the success of tree-based ensemble techniques is known to be dependent on the form taken by the predictive model representing each leaf node and the diversity among the tree models comprising the ensemble.

With respect to the form of the predictive model at each leaf node, the popular CART [4], Bayesian CART [8, 9, 20], and BART [7] methods have demonstrated remarkable success in a variety of settings,

typically using simple scalar responses at each leaf node (i.e., $f(x_t; T_i, \theta_i) = \mu_{i,p_i(x_t)}$). Gramacy and Lee [12] and Broderick and Gramacy [5] have extended these approaches by using linear or Gaussian process models at the leaf nodes (i.e., $f(x_t; T_i, \theta_i) = \beta'_{i,p_i(x_t)} x_t$ in the linear case). Although we focus on *predictive* models in this paper, it is worth noting that for classification problems, many popular and powerful additive tree classifiers involve the use of *discriminatory* methods, e.g., linear discriminant analysis, at each node [14].

The diversity among the tree models comprising an ensemble can be viewed as a function of the mode of inference, and on this point the literature on additive tree methods is divided. The two foremost approaches for conducting inference are those derived from the Random Forest technique proposed by Breiman [3] and those derived from formal Bayesian probability theory [6, 8, 20]. It is worth recognizing that, although there are profound philosophical differences between these two approaches, in both cases the goal is fundamentally the same – to increase the predictive strength of the ensemble by averaging over uncertainty in the underlying tree models. The practical difference between the two lies in the method employed to quantify that uncertainty. In the Random Forest approach, uncertainty in the component tree models is introduced via bagging and feature selection. In the Bayesian setting, uncertainty arises naturally by sampling from the joint posterior distribution. It seems to be generally accepted that the preference for the Bayesian approach (when such a preference exists!) is due to the greater interpretability of the uncertainty measures. It may be distressing [to Bayesians] then, that fully Bayesian approaches are often shown to have slightly, but significantly, inferior prediction accuracy when compared to Random Forests for classification tasks [7, 1, 21]. One of the objectives of this paper is to begin to unify these two approaches, if only in the context of additive tree classifier models, to exploit the desirable properties of both. We will return to this topic again in Section

6.

3 Classification Using Additive Logistic Ensembles

Since our primary interest is in classification, we begin with an additive tree-based ensemble model based on the logistic regression formulation presented in Section 2. Although the approach may be generalized and applied to multi-state classification problems, for ease of notation we only consider the two-class version here.

3.1 Overview

Our two-class classification model is as follows[1]:

$$p(y_t = 1|\eta_t) = \text{logit}^{-1}(\eta_t) \quad (1)$$

$$= \frac{1}{1+\exp(-\eta_t)} \quad (2)$$

$$\eta_t|x_t = \sum_{i=1}^{M} \beta'_{i,p_i(x_t)} x_t \quad (3)$$

For each component model, the vector of features is mapped to a specific leaf node, i.e., $p_i(x)$, through a series of binary comparisons at each interior (or *split*) node of a binary decision tree T_i. More specifically, a split feature $x_{t,k}$ is selected (from among K features) and compared to a threshold r at each node. If the feature value is smaller than the threshold, the sample is sent to the left, and otherwise, it is sent to the right. This process is repeated until the sample finally arrives at a leaf node. Each decision tree T_i is therefore

[1]Though other BART-style ensemble classifiers [7, 21, 1] favor the probit function, we chose to use the logistic link function due to its ability to represent the full posterior density in closed form. This is required for our MAP search heuristic.

characterized by its topology, along with the split feature and threshold for each split node. Each leaf node is, in turn, characterized by the regression coefficients given by $\beta_{i,p_i(x_t)}$.

We have elected to denote the parametric model associated with each leaf node as a multivariate regression involving the entire feature vector x_t. In practice, however, we use a univariate model at each leaf node, involving only the single feature used as the split feature in the penultimate node. Our univariate regression provides a measure of distance from the final decision boundary and incurs far less computational overhead than the full regression.[2]

3.2 Prior Distributions

To complete our model specification, we assign prior distributions to the free parameters in the likelihood described in the previous section. Note that each tree-based regression model in the ensemble is denoted as $T_i = \{s_i, r_i, \beta_i\}$ for $i = 1 \ldots M$ and contains the following parameters: a set of split nodes $s_i = \{s_{i,1} \ldots s_{i,|s_i|}\}$ which determine the feature involved in each split (i.e., $s_{i,j} \in \{1 \ldots K\}$ for $j = 1 \ldots |s_i|$); the corresponding set of thresholds $r_i = \{r_{i,1}, \ldots, r_{i,|s_i|}\}$ associated with the split nodes; and a set of regression coefficients $\beta_i = \{\beta_{i,1}, \ldots, \beta_{i,|\beta_i|}\}$ associated with each leaf node.

We will assume (like Chipman et al [8], Wu et al [20], Gramacy and Lee [12], and others) a set of conditional independencies in the prior distribution such that the full prior distribution factors as

$$p\left(\{T_i\}_{i=1}^{M}\right) = p(M) \prod_{i=1}^{M} \left(p(|s_i|) \prod_{j=1}^{|s_i|} p(r_{i,j}|s_{i,j}) p(s_{i,j}) \prod_{l=1}^{|\beta_i|} p(\beta_{i,l}) \right). \tag{4}$$

We have assumed that all trees sharing a particular number of split nodes (and therefore the same number of leaf nodes) have the same prior probability. Hence, the prior distribution for the topology of a tree can be

[2] An alternative method for increasing computational efficiency involves using a reduced multivariate regression model that incorporates solely those features utilized in the splits at the interior nodes.

expressed as a prior over its number of split nodes. This will be justified below.

In general, we will attempt to choose priors that penalize the memory requirements of the ensemble *in silico*. In the following section we present a novel set of prior distributions fundamentally derived from the principal of minimum description length (MDL) [16], an appropriate motivation given that a primary objective of this work is to find Bayesian additive tree ensembles with minimal memory complexity. Note that our MDL arguments will often be presented in terms of penalizing the storage requirements of larger models. We can regard these penalization formulae as proportional to probability density functions. The penalization induced by the MDL-motivated priors is usually interpreted in terms of the $\log_2$ of the density assigned by the prior. This is appropriate, since the posterior optimization (where the role of the priors is one of regularization) will occur in log space.

3.2.1 Number of Trees

We begin with a flat prior over the number of trees, i.e.,

$$p(M) \propto 1. \tag{5}$$

In practice, this means that we ignore the size of the ensemble while fitting. However, each tree added to the ensemble will decrease the prior component of the joint posterior, because each new model will necessarily involve split nodes, thresholds, and leaf node parameters. Although the prior as specified is improper, we use it here for simplicity. A discrete uniform distribution on positive integers would produce equivalent results, but our notation eliminates the need to specify an upper bound on the size of the ensemble. The improper formulation can also be easily justified from the MDL perspective. That is, when storing the parameters associated with the model, the storage requirement will be derived from the parameters associated with each

component of the ensemble and not from the size of the ensemble itself.

3.2.2 Number of Split Nodes

Tree shape priors were pioneered by Buntine [6] and Chipman et al [8] and continued by [20] and Gramacy and Lee [12]. These approaches incorporate several different parameters that influence the probability that a given decision tree node is a leaf. Unfortunately, priors structured in this fashion often require complete knowledge of the global tree structure in order to properly alter the tree, and therefore they entail more processing time to perform each alteration. In order to conduct inference on the tree topology in a more computationally efficient way, it would be convenient if the prior possessed a more limited *local* structure. This would effectively restrict the burden of carrying out alterations to subtrees alone and allow the prior for the entire tree to be more quickly assessed.

An alternative approach to this problem would be to assume a flat prior on tree topologies. Using this strategy, the prior probability of a tree will not decrease as a direct result of adding a node[3]. On the other hand, a flat prior cannot be justified from an MDL standpoint, because at least some memory is required to store the structure of the tree. Note that memory is not required to store the leaf nodes – their proper connections can be inferred directly from knowledge of the split nodes. Therefore, to represent the tree, we simply need a trinary feature for each split node that tells us whether it is a parent to 0, 1, or 2 other split nodes. Under this coding scheme, the prior should penalize each additional split node by $\log(3)$, the number of bits required to store the associated information. Thus, we can assume the prior probability of the tree

[3]However, the addition of a node will require the addition of split features, split thresholds, and regression coefficients *associated* with that node, ultimately driving down the tree probability.

topology to be given by

$$p\left(|s_i|\right) \propto 3^{-|s_i|} \tag{6}$$

where $|s_i|$ is the random feature representing the number of split nodes.

3.2.3 Split Features and Thresholds

As described in Section 3.1, each observation is mapped to a unique leaf node using a binary decision tree. For each split node of the tree structure, the decision involves determining whether the observation should move to the left or right child node based upon a threshold value for a particular element of the feature vector (i.e., the value associated with the corresponding split feature). Hence, prior distributions are required for (1) the conditional distributions of the split features given the tree structure, and (2) the threshold values given the split features. We first assume that each of the K features is equally likely to serve as the split feature for each split node of the tree, i.e.,

$$p\left(s_{i,j}\right) = \frac{1}{K}. \tag{7}$$

Once again, this prior is intuitively justifiable from the MDL perspective, because $\log K$ bits are required to store the index of the feature being used as the split feature. This prior implies a penalization of $\log K$ per split node.

For each split feature, we must also select a threshold value that determines how the data at the interior node are to be split. We make the observation that in practice, the observed data can only provide evidence for threshold values between the realized values of the associated feature. For this reason, we assume an empirical prior distribution, dependent on the data, that assigns equal probability to all candidate thresholds

falling halfway between each pair of observed data points. The distribution can be written as

$$p\left(r_{i,j}|s_{i,j}\right) = \frac{1}{N_r(x_{s_{i,j}})}. \tag{8}$$

Here $N_r(x_{s_{i,j}})$ is the number of thresholds, or unique pairs of values, associated with the feature $x_{s_{i,j}}$. Although this is not a strictly valid prior (because it is data dependent), it has yielded stronger performance than the traditional data independent priors we have investigated. An interesting property of this prior is that it will penalize the use of features with many possible thresholds, thereby favoring features for which the decision problem is easier.

3.2.4 Leaf Node Regression Coefficients

We assume a standard mean 0 multivariate Gaussian prior for the vector of regression coefficients associated with each leaf node, p, given by

$$p\left(\beta_{i,p}\right) \propto (2\pi)^{-d/2}|\Sigma|^{-1/2}\exp\left(-\frac{1}{2}\beta'_{i,p}\Sigma^{-1}\beta_{i,p}\right) \tag{9}$$

where d is the dimensionality of the linear parameters chosen. In practice, we choose $\Sigma = \text{diag}(\sigma)$ with a large value of σ such that any single leaf node could saturate the logistic function. For example, if the features are normalized to have mean 0 and standard deviation of 1, we set σ to be 10.

Finally, incorporating all of these terms, the full prior is given by

$$p\left(\{T_i\}_{i=1}^{M}\right) \propto \prod_{i=1}^{M}\left(3^{-|s_i|}\prod_{j=1}^{|s_i|}\frac{1}{KN_r(x_{s_{i,j}})}\prod_{l=1}^{|\beta_i|}(2\pi)^{-d/2}|\Sigma|^{-1/2}\exp\left(-\frac{1}{2}\beta'_{i,l}\Sigma^{-1}\beta_{i,l}\right)\right). \tag{10}$$

4 Greedy Search for Posterior Optimization

A MAP estimate is usually obtained by maximizing the log of the posterior. However, in this paper, we minimize the negative of the log of the posterior and refer to it as the error, E. We refer to the term due to the log-likelihood as E_{data} and the term due to the prior as E_{prior}, where

$$E = -\log\left(P\left(y_{1..N}|x_{1..N}, Ensemble\right)\right) - \log\left(P\left(Ensemble\right)\right) = E_{data} + E_{prior}. \tag{11}$$

To implement node splitting in many other decision tree fitting methodologies, a given function is evaluated at many or all of the candidate threshold values. The threshold value that optimizes that function is selected to split the node, effectively determining which samples move into the left or right child node. We use a similar approach, in which the function to be optimized is an approximation to the error – that is, we must find the minimum error for a given feature and threshold.

The algorithm we derive is simlar in nature to the LogitBoost algorithm. The main differences are due to the addition of the model prior and the maximization of the posterior rather than the logit loss function. A derivation of the algorithm, including the mathematical underpinnings that show how this maximizes the posterior, is provided in the Appendix.

4.1 Determining Tree Architecture with Boosting

To perform boosting, we must first compute the weight w_t and the load z_t for each sample. Roughly speaking, the weights w_t measure the uncertainty of the current ensemble in the class of the sample and hence, the degree to which the next learner (i.e., decision tree) should focus on that sample. The loads z_t are a measure of which class the next learner should push the tth data sample towards. They are calculated as:

$$w_t = \text{logit}^{-1}(\eta_t)\left(1 - \text{logit}^{-1}(\eta_t)\right), \tag{12}$$

and

$$z_t = \frac{y_t - \text{logit}^{-1}(\eta_t)}{\text{logit}^{-1}(\eta_t)\left(1 - \text{logit}^{-1}(\eta_t)\right)}. \tag{13}$$

Initially, when the ensemble is empty, all the $\text{logit}^{-1}(\eta_t)$ are assumed to be 0.5. This is equivalent to assuming an ensemble having zeros for all the activations. These values are recalculated for use by each learner.

To determine whether (and how) a node should be split, we place a threshold between every pair of observed data points and update two sufficient statistics for each of the child nodes based on their respective data. The two statistics employed are very similar to those required for linear regression, i.e.,

$$A = \sum_t^N w_t x_t x_t' \tag{14}$$

and

$$b = \sum_t^N w_t x_t z_t \tag{15}$$

where t now indexes the set of data in a given child node. The effective error of each child node can be calculated from the sufficient statistics as (see appendix for derivation)

$$E_{left} = -\frac{1}{2} b'_{left} \left(A_{left} + \frac{1}{\sigma^2} I \right)^{-1} b_{left} - \frac{d}{2} log\left(2\pi\sigma^2\right), \tag{16}$$

$$E_{right} = -\frac{1}{2} b'_{right} \left(A_{right} + \frac{1}{\sigma^2} I \right)^{-1} b_{right} - \frac{d}{2} log\left(2\pi\sigma^2\right), \tag{17}$$

$$E_{split} = E_{left} + E_{right} + log\left(N_f\right) + log\left(N_t\right) + log\left(3\right). \tag{18}$$

Note that the identity matrix ensures that the matrix is always invertible. However, because this matrix inversion must be performed for every possible threshold, a high dimensional linear model at the leaf nodes can slow down training considerably. Here, we use only the feature dimension that the parent node used as

the splitting feature. Using this dimension as a constant in our linear model yields a rank 2 matrix that must be inverted for every threshold. This can be performed very quickly for such a small matrix.

We then choose the feature and threshold with the lowest error for the left and right child nodes. This lowest error is compared with the error incurred by not splitting, incorporating the penalties imposed by the prior for adding a split node.

$$E_{nosplit} = -\frac{1}{2}b'\left(A+\frac{1}{\sigma^2}I\right)^{-1}b - \frac{d}{2}log\left(2\pi\sigma^2\right) \tag{19}$$

$$E_{split} <=> E_{nosplit}. \tag{20}$$

If splitting increases the error, the split is not carried out, and the splitting stops. In the next section we will discuss determination of the optimal linear parameters.

4.2 Determining Linear Parameters

Up to this point, the tree architecture was determined using a local squared error approximation to the likelihood. This approximation enables fast calculation of the errors, which is necessary to investigate all the feature and threshold values. If the LogitBoost algorithm were followed here, the linear parameters that minimized the same approximation to the error would be used. This optimal value of linear parameters is calculated from the same sufficient statistics as (see Appendix for derivation)

$$\beta_{MAP} = \left(A+\frac{1}{\sigma^2}I\right)^{-1}. \tag{21}$$

However, we have observed that naively following the LogitBoost algorithm and fitting the linear parameters to the squared error approximation causes some trees to increase the true value of E_{data}. This is because the approximation is only valid locally, and greedily optimizing parameters with respect to the approximation can drastically change parameter values. Clearly, adding a tree to the ensemble that increases E_{data}

decreases the posterior probability of the ensemble. Using β_{MAP} as the estimate of the linear parameters is equivalent to taking a Newton-Ralphson step in error space. The error increases because our step is too large. This is the reason most boosting algorithms add so many learners without overfitting the data. Some learners increase the error by taking too large of a step and the other learners remove this error. Our methodology improves upon boosting by actually finding the linear parameters that minimize the true error before advancing to the next tree. We accomplish this by using a version of iteratively reweighted least squares, stopping if the value hasn't converged after 5000 iterations. This approach results in very few trees and adds no free parameters.

Once splitting of the first tree has stopped, it is added to the ensemble. The weights (w_t) and loads (z_t) are updated and sent to the next tree to continue the process. In general boosting, this would continue until the user decides to stop or until some heuristic stopping criterion is reached. However, in CATE, as more of the likelihood is explained by additional trees, less complexity is justified in future trees; hence, the trees get progressively smaller. Once CATE finally produces a tree with one node, training stops. Thus, the algorithm automatically determines the number of trees in the ensemble.

5 Results

To illustrate the strengths of the Compact Additive Tree Ensemble (CATE) method, we compare its performance to that of Bayesian Additive Regression Trees (BART) and the Random Forest (RF) as described by Breiman [3]. Our goal is to show that the compact ensemble achieves similar performance to other methodologies in terms of accuracy while producing a smaller ensemble that is significantly faster to train and classify. Because a smaller and faster classifier is likely to be more advantageous for larger data sets, we

selected several large, two-class classification data sets from the UCI Machine Learning Repository [10]. These data sets include the Magic Gamma Ray Telescope data set [2], the Adult (Census Income) data set [13], the Bank Marketing data set [17], and data taken from the URL Reputation data set [15]. For each experiment, where applicable, we have reported the individual class accuracies, average class accuracy, training and classification time, average tree size, forest size and the memory footprint of the model. The parameters used for training each of the models are provided in Section 5.1.

5.1 Training the Models

The RF requires that at least two parameters be specified, namely, the number of trees and the number of features to sample randomly at each split node. The latter is referred to as the split dimension, and varies with data set. In our experiments, we have simply selected a split dimension for each of these experiments that is consistent with the heuristic recommended by Breiman [3] – the square root of the total number of features. In each experiment, the RF model was run with 200 trees, as this appeared (via limited empirical testing) to be close to its saturation point on these data.

For the BART experiments, we employed the R implementation found in the *BayesTree* package[4]. Because the BART model is a fully Bayesian model of an ensemble of trees, the number of trees in the ensemble must be chosen, along with the number of ensembles to sample from the posterior distribution. For these experiments, we have chosen 10 trees for the ensembles, and 200 samples from the posterior. All other parameters were set to their default values.

As for CATE, one of its premier advantages is that the appropriate number of trees is determined auto-

[4] Average tree size and memory footprint are not readily isolated in the R implementation of BART, so these measurements are not provided for this algorithm.

matically. Other than the default prior values discussed in previous sections, no parameters are specified a priori for CATE.

5.2 Magic Gamma Ray Telescope Data Set

The Magic Data Set consists of 12,281 training samples and 6,339 testing samples associated with 10 real-valued features. Class 1 samples comprise roughly two-thirds of the data (with the remaining one-third of class 0). A summary of these results are provided in Table 1. We observe that, arguably due to data imbalance, all three methods performed much more poorly on class 0 data than on class 1. Notably, the performance of CATE (in terms of average accuracy) is consistent with that of the other methods. More importantly, CATE produces a model with only 6 trees, yielding a much smaller memory footprint and considerably faster training and classification.

Method	**Accuracy**		**Avg.**	**Time (s)**		**Forest Size**	**Avg. Nodes**	**Memory (MB)**
	Class 0	**Class 1**	**Accuracy**	**Training**	**Testing**			
CATE	0.7074	0.9344	0.8209	0.341	0.004	6	15.7	0.002
RF	0.7743	0.9401	0.8572	9	1	200	2327.1	11.2
BART	0.7092	0.9271	0.8181	12	1	10	NA	NA

Table 1: Magic Data

5.3 Bank Marketing Data Set

The Bank Marketing Data Set consists of 45,211 samples associated with 16 features. A random sample of instances, half the size of the original data set, was selected for testing. Note that approximately 88% of the data consists of class 0 samples, while the remainder are of class 1. Ten features were used in this

experiment, including only the real-valued features and the two-valued categorical features. A summary of these results are provided in Table 2. As we observed in the Magic data experiments, this moderate data imbalance appears to adversely affect the performance of all three algorithm on class 1 data. This aside, CATE performs as well as its competitors. Again, it produced a 6-tree forest and remarkably faster training and classification.

Method	Accuracy		Avg.	Time (s)		Forest Size	Avg. Nodes	Memory (MB)
	Class 0	Class 1	Accuracy	Training	Testing			
CATE	0.9667	0.3272	0.6470	0.415	0.015	6	12	0.001
RF	0.9736	0.3148	0.6442	8	3	200	2743.5	13.2
BART	0.97244	0.2994	0.6359	22	3	10	NA	NA

Table 2: Bank Data

5.4 URL Reputation Data Set

The URL Reputation data set is quite large, consisting of 2.4 million samples and 3.2 million features spread across 121 days of data. For our experiments, we used days 0 through 59 as the training set and days 60 through 120 as the testing set. This results in 1,176,130 training samples and 1,200,000 testing samples. Unfortunately, the number of dimensions makes it unfeasible to search every dimension and every associated data point for threshold values. Moreover, due to memory limitations in the R framework, BART crashed under the weight of this entire data set. To mitigate these issues, we used only the 64 continuous real features, and we performed 20 runs, each on a randomly selected 10% of the training and testing sets[5].

[5]Conceivably, an even smaller number of features may be effective if combined with an efficient feature selection strategy. This may be explored in future work.

The performance values, averaged across all 20 runs, are provided in Table 3. For the sake of completeness, we have also included the performance of CATE and the RF on the entirety of URL data.

Despite the size of this data set, we observe immediately that on average CATE still produces a very small model (13 trees) consisting of very small trees. There is no doubt that this characteristic of CATE contributes largely to its more efficient performance. Moreover, these results suggest that CATE scales to larger data sets well and achieves comparable performance to the RF and BART.

Method	**Dataset**	**Accuracy**		**Avg.**	**Time (s)**		**Forest**	**Avg.**	**Memory**
	Size	**Class 0**	**Class 1**	**Accuracy**	**Training**	**Testing**	**Size**	**Nodes**	**(MB)**
CATE	100%	0.9835	0.9252	0.9543	919	1.8	13	400.1	0.095
RF	100%	0.9900	0.9244	0.9572	2694	348	200	40517	194.5
CATE	10%	0.9755	0.9153	0.9454	14.6	0.113	9.2	72.4	.011
RF	10%	0.9865	0.9190	0.9528	147	22.1	200	7585.3	36.4
BART	10%	0.9455	0.7906	0.8681	202.8	21.7	10	NA	NA

Table 3: URL Data

5.5 Adult Census Income Data Set

The Adult Data Set consists of 32,561 training samples and 16,281 testing samples associated with 14 features. Only the 6 real-valued features were used in our experiments. Class 0 samples comprise roughly 75% of the data (and the remainder are of class 1). Note that the above experiments essentially tell the same story, i.e., that CATE achieves comparable performance to the RF, while yielding a smaller model and more efficient training and classification. However, one may conjecture that restricting tree depth, particularly in the case of the RF, might adequately control model size and efficiency, thereby leveling the playing field. We have tested this conjecture for tree depth limits 10 and 3 and found that this is not the case. A summary

of these results is provided in Table 4. Although the performance on Class 1 is poor across the board, the performance of the RF under depth constraints clearly falls apart. On the other hand, even a maximum tree depth of 3 did not significantly impact the performance of CATE. This parameter did increase the forest size produced by CATE, suggesting that its individual trees might be less effective individually. However, the forest remained small, efficient and highly effective.

Method	**Max.**	**Accuracy**		**Avg.**	**Time (ms)**		**Forest**	**Avg.**	**Memory**
	Depth	**Class 0**	**Class 1**	**Accuracy**	**Training**	**Testing**	**Size**	**Nodes**	**(MB)**
CATE	None	0.9538	0.4748	0.7143	0.647	.011	6	15	0.002
RF	None	0.9548	0.4574	0.7061	10	3	200	3674.9	17.6
BART	None	0.9573	0.4212	0.6893	37	4	10	NA	NA
CATE	10	0.9538	0.4745	0.7142	.647	.011	6	15	0.002
RF	10	0.9987	0.2301	0.6144	7	1	200	527.2	2.5
CATE	3	0.9591	0.4623	0.7107	.826	.011	10	5.2	0.001
RF	3	1	0	0.5	2	0.267	200	14.7	0.072

Table 4: Adult Data with Maximum Tree Depth

As one might expect when testing multiple methodologies on several different data sets, there is no clear ”winner” among these three methods strictly in terms of accuracy. Indeed, class imbalance appears to be a stronger predictor of performance in these experiments than the classification method itself. As hoped, CATE performs comparably to the other methods. However, it is in the training time and the compactness of the overall model where CATE greatly outperforms its competitors.

6 Discussion

We have shown that the Compact Additive Tree Ensemble method produces compact simple ensembles with significantly fewer (and less complex) trees than the Random Forest methods. We have also shown

that the performance of CATE, in terms of class average accuracy, is comparable with RF and BART on four different, publicly available data sets. It is important to note that, in keeping with expected operating conditions, we assumed little a priori knowledge about the feature data aside from a handful of empirical tests. Hence, the split dimension, ensemble size and other parameters used by the RF and BART methods were selected somewhat naively. The impact of this is that we cannot readily determine whether these methods are being optimally applied, which would likely be the case if they were deployed in the field. CATE, on the other hand, relies only upon default prior parameters, requiring no input from the user. We have found empirically that these default values perform very well across a range of problems, but like other models, they could conceivably be tuned to a specific problem if small differences in accuracy were important for a given application.

In addition to its compactness, its comparable performance to Random Forests, and its automatic determination of the number of trees, CATE has the advantage of being cast in a rigorous Bayesian framework. This enables essentially any priors used in other Bayesian tree work to be introduced and any of the tools used with logistic regression to be employed. A key difference between CATE and the probit BART classifiers previously described [7, 21, 1] is our use of partition-specific linear models. We model the latent η features using an ensemble of treed regression models similar to those described by Chipman, et al. [9]. This represents a slightly more complex leaf node model than typically favored for BART-style ensembles. However, since one of the major goals of our approach is to produce a compact ensemble, the greater complexity of this choice of leaf node model has negligible effects on computational efficiency due to the substantially decreased size of the ensemble. Note also that methods such as that of Gramacy and Lee [12] – in which the leaf node models may be either linear (as here) or Gaussian processes – while not expressly ensembles, are *effectively* additive ensembles in that their predictions are generated using additive combinations of the

component models contained in the posterior sample obtained by the Monte Carlo method.

From an algorithmic point of view, our optimization technique is most similar to Logit Boosting C4.5 trees [18]. During training, our model is largely likelihood driven (similar to information metrics), and early stopping has been attempted with C4.5. However, in CATE the number of trees is self-determined. Also, the greedy MAP Forest ensures that each tree will reduce the likelihood, unlike the typical LogitBoost algorithm [11]. Note that these techniques are not guaranteed to find the true MAP. First of all, the problem of finding the tree architecture that minimizes the error is not convex, and hence we may, in fact, be finding local maxima. Secondly, we stop when no more trees can be justified by the data rather than when the model has converged completely. To find the true MAP estimate we would need to remove the first tree and fit another in its place, repeating the process until the new tree matches the one removed. However, following this procedure would require additional training time and would possibly lead to overfitting.

We believe our approach is ideal in many ways for BART-style ensembles, as well as for constructing explicit ensembles from other flavors of treed models. It will be interesting to compare the performance of the ensembles produced by our method to the performance of the exact same underlying model as fitted by MCMC.

We believe this general approach constitutes a valuable addition to the body of work surrounding Bayesian additive tree models. It retains the accuracy of Random Forest methods and is even simpler to use than Random Forests and many Bayesian methods, which often require a number of input/hyper-parameters. Our approach also retains the interpretability of Bayesian methods, while offering excellent computational efficiency during both training and testing. This is particularly true in the case of large data sets such as the URL Reputation, where other methods may prove to be impractical.

Ultimately, for large enough data sets we believe that CATE will not only be beneficial, but will prove

to be indispensible for applications requiring rapid inference in a classification framework.

References

[1] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair. Bayesian additive regression trees-based spam detection for enhanced email privacy. In *Third International Conference on Availability, Reliability, and Security*. IEEE, 2008.

[2] RK Bock, A. Chilingarian, M. Gaug, F. Hakl, T. Hengstebeck, M. Jiina, J. Klaschka, E. Kotr, P. Savickı, S. Towers, et al. Methods for multidimensional event classification: a case study using images from a Cherenkov gamma-ray telescope. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 516(2-3):511–528, 2004.

[3] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[4] L. Breiman, J. H. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.

[5] T. Broderick and R. B. Gramacy. Classification and categorical inputs with treed gaussian process models. *Journal of Classification*, 28(2):244–270, 2011.

[6] W. Buntine. Learning classification trees. *Statistics and Computing*, 2(2):63–73, 1992.

[7] H. A. Chipman, E. I. George, and R. E. McCulloch. BART: Bayesian additive regression trees. *Annals of Applied Statistics*, 4(1):266–298, 2010.

[8] H.A. Chipman, E.I. George, and R.E. McCulloch. Bayesian CART model search. *Journal of the American Statistical Association*, 93(443):935–948, 1998.

[9] H.A. Chipman, E.I. George, and R.E. McCulloch. Bayesian treed models. *Machine Learning*, 48(1):299–320, 2002.

[10] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[11] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Machine learning: Proceedings of the thirteenth international conference*, pages 148–156, 1996.

[12] R. B. Gramacy and H. K. H. Lee. Bayesian treed Gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103(483):1119–1130, 2008.

[13] R. Kohavi. Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD, 1996.

[14] T. D. Lemmond, B. Y. Chen, A. O. Hatch, and W. G. Hanley. An extended study of the discriminant random forest. In *Data Mining*, pages 123–146. Springer, 2010.

[15] J. Ma, L.K. Saul, S. Savage, and G.M. Voelker. Identifying suspicious URLs: an application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 681–688. ACM, 2009.

[16] D. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, Cambridge, UK, 2003.

[17] S. Moro, R. Laureano, and P. Cortez. Using Data Mining for Bank Direct Marketing: An Application of the CRISP-DM Methodology. In *Proceedings of the European Simulation and Modelling Conference*, pages 117–121. ESM, 2011.

[18] J.R. Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the National Conference on Artificial Intelligence*, pages 725–730, 1996.

[19] B. Van Essen, C. Macaraeg, M. , Gokhale, and R. Prenger. Accelerating a random forest classifier: multi-core, GP-GPU, or FPGA? In *IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pages 232–239. FCCM, 2012.

[20] Y. Wu, H. Tjelmeland, and M. West. Bayesian CART. *Journal of Computational and Graphical Statistics*, 16(1):44–66, 2007.

[21] J. L. Zhang and W. K. Härdle. The Bayesian additive classification tree applied to credit risk modelling. *Computational Statistics and Data Analysis*, 54:1197–1205, 2010.

A Boosting Details

Our approach for determining whether and how to split data at frontier nodes in the newest tree is similar in spirit to the approach taken by Logit Boost [11]. In particular, our algorithm adds split nodes to the newest tree in the ensemble that help to minimize the logit loss, but unlike Logit Boost, our approach adds an additional term to the loss function that penalizes for additional model complexity. For a given sample x_t, the overall loss is given by:

$$E\left(\sum_{i=1}^{M} a_{ti}\right) = E_{data_t}\left(\sum_{i=1}^{M} a_{ti}\right) + E_{prior}\left(\sum_{i=1}^{M} a_{ti}\right) \tag{22}$$

where, for simplicity of notation, we define the activation value generated by the ith tree for x_t as:

$$a_{ti} = \beta'_{i,p_i(x_t)} x_t \tag{23}$$

and $E_{data_t}()$ and $E_{prior}()$ are the error terms that penalize for lack of fit and model complexity respectively and will be defined shortly.

$E_{data_t}()$ is the standard logit loss function that we will write as:

$$E_{data_t}\left(\sum_{i=1}^{M} a_{ti}\right) = -y_t \log\left(\text{logit}^{-1}\left(\sum_{i=1}^{M} a_{ti}\right)\right) + (y_t - 1)\log\left(1 - \text{logit}^{-1}\left(\sum_{i=1}^{M} a_{ti}\right)\right) \tag{24}$$

for y_t taking on the values zero or one.

Each time a new tree is added, a new i is added to the activation sum. We can Taylor expand the E_{data} term about the activation of the new tree which we refer to as a_M. For brevity, we use the notation:

$$\sum_{i=1}^{M-1} a_{ti} = a_{old}. \tag{25}$$

The expansion to second order is given by:

$$E_{data_t}\left(a_{old} + a_M\right) \approx E_{data_t}\left(a_{old}\right) + \frac{\partial E_{data_t}\left(a_{old}\right)}{\partial a_M} a_M + \frac{1}{2}\frac{\partial^2 E_{data_t}\left(a_{old}\right)}{\partial a_M^2} a_M^2. \tag{26}$$

Using the first and second derivatives of the logit loss with respect to the activations, i.e.,

$$\frac{\partial E_{data_t}\left(a_{old}\right)}{\partial a_M} = \text{logit}^{-1}\left(a_{old}\right) - y_t, \tag{27}$$

and

$$\frac{\partial^2 E_{data_t}\left(a_{old}\right)}{\partial a_M^2} = \text{logit}^{-1}\left(a_{old}\right)\left(1 - \text{logit}^{-1}\left(a_{old}\right)\right) \tag{28}$$

to write the difference in E_{data} due to the new activation as

$$E_{data_t}\left(a_{old} + a_M\right) - E_{data_t}\left(a_{old}\right) \approx \frac{w_t}{2}\left(a_M - z_t\right)^2 + \frac{w_t z_t^2}{2} \tag{29}$$

where the last term on the right does not depend on a_M and

$$w_t = \text{logit}^{-1}\left(a_{old}\right)\left(1 - \text{logit}^{-1}\left(a_{old}\right)\right), \tag{30}$$

and

$$z_t = \frac{y_t - \text{logit}^{-1}\left(a_{old}\right)}{\text{logit}^{-1}\left(a_{old}\right)\left(1 - \text{logit}^{-1}\left(a_{old}\right)\right)}. \tag{31}$$

The terms w_t and z_t are referred to as the "weights" and the "loads" respectively in the boosting literature [11]. As each new learner is added, it is trained using the weights and load from the previous learner to optimize the overall cost function.

As shown in (3) in section 3.1 the activations are generated from the linear parameters and the input features. The linear parameters at each of the leaf nodes are sampled independently. This means that new activations add a term to the E_{prior}

$$E_{prior}\left(a_{old} + a_M\right) = E_{prior}\left(a_{old}\right) + E_{prior}\left(a_M\right) \tag{32}$$

and the change in prior due to the activation is just the negative log of the Gaussian

$$E_{prior}\left(a_{old}+a_M\right)-E_{prior}\left(a_{old}\right)=\frac{{\beta_M}'\beta_M}{2\sigma^2}-\frac{d}{2}log\left(2\pi\sigma^2\right) \tag{33}$$

Using (3) in section 3.1 it's clear that finding the optimal a_M means finding the optimal linear parameters and because each of the linear parameters on the leaf nodes are assumed to be independently sampled, the penalty for the complexity of new linear parameters in E_{prior} is added to the old.

Because each of the samples are assumed to be independent, the E_{data} terms for each sample are summed. We now need to find the value of a_M that minimizes this squared error, and the prior term. Thus it our expansion can be written

$$E\left(a_{old}+a_M\right)-E\left(a_{old}\right)\approx\sum_{t=1}^{N}\frac{w_t}{2}\left(\beta_M'x_t-z_t\right)^2+\frac{w_tz_t^2}{2}+\frac{{\beta_M}'\beta_M}{2\sigma^2}-\frac{d}{2}log\left(2\pi\sigma^2\right) \tag{34}$$

where the constant doesn't depend on β_M. For simplicity we will refer to this expansion of the error as E_{new}. Thus

$$E_{new}=\sum_{t=1}^{N}\frac{w_t}{2}\left(\beta_M'x_t-z_t\right)^2+\frac{{\beta_M}'\beta_M}{2\sigma^2}+constant \tag{35}$$

This is the same cost function as that is used in ridge regression and the optimal value can be written in terms of two sufficient statistics, A and b

$$\underset{\beta_M}{\text{argmax}}E_{new}=\left(A+\frac{1}{\sigma^2}I\right)^{-1}b \tag{36}$$

where

$$A=\sum_{t=1}^{N}w_tx_tx_t' \tag{37}$$

and

$$b = \sum_{t=1}^{N} w_t x_t z_t. \tag{38}$$

Here I is the identity matrix. The identity matrix from our spherical prior ensures that the matrix is always invertible. The expansion of the error simplifies when it is evaluated at this optimal value of β_M, which we will refer to as $\hat{\beta}$

$$E_{new}|^{\hat{\beta}} = -\frac{1}{2} b' \left(A + \frac{1}{\sigma^2} I \right)^{-1} b \tag{39}$$